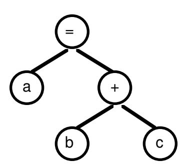
Life Story of an Add Instruction

As a summary for Chapter 1, and an attempt to put the various components of instruction execution into context, the following is the life story of an add instruction.

Long ago and far away ...

Programmer writes a = b + c
Compiler parses statement



Compiler generates assembly code

```
lw  $8,b       ; fetch b value
lw  $9,c       ; fetch c value
add $10,$9,$8  ; form sum
sw  $10,a      ; save result
```

Conversion of assembly code

Variables assigned offsets in global area

```
a <—> $28+64
b <—> $28+68
c <—> $28+72
```

Instructions converted to binary

`add  $10,$9,$8 <-->` op=0, rs=10, rd=9, shamt=0, func=32

```
000000 01000 01010 01001 00000 100000
```

Program, data and other segments written to disk
OS sets up segment table, establishes User ID
Page table entry set up for each page Disk Addr + invalid

OS Fires Things Up

- Segment table base = UID field, PC field = addr 1^{st} inst
- PSW "installed" to cause execution of pgm to start
- TLB faults
- Virtual-to-phys. mapping computed, TLB set, but invalid
- Page fault, page frame allocated, disk read initiated ...
- On arrival, valid bit set in page table & TLB, pgm restarted
- Ref. hits in TLB, but misses in cache; cache line filled

Executing the Add

- ... After `lw $9,c` "completed", PC points to add
- PC presented to TLB, hits producing physical addr
- Phys. addr presented to cache, hits yielding instr.
- Instruction moved to the IR
- Decoder determines its `add`, selects operand registers
- ALU control signals generated from func field
- Two operands and control signals presented to ALU
- In each bit position, true/compl operands and carry-in are inputs to an adder producing result and carry-out
- <Details on gates, circuits, field effect transistors>
- OR of all 32 result bits & bit_{31} carry out yield 4-bit Cond. Code
- Result assigned to destination register
- PC incremented by 4